Senior Design Projects

Interpretable AI and LLM Safety Deployments

Spring 2026, University of Florida

Faculty Advisor: Dr. Sumit Kumar Jha

https://sumitkumarjha.com/

IMPORTANT: These are deployment projects, not research projects.

Your job is to implement, optimize, and deploy existing methods, not invent new ones.
You can request access to the original papers to help you succeed.

DRAFT

Contact: sumit.jha@ufl.edu

Project 1: HETA Lite

Real Time Token Attribution for Decoder Only LLMs

Based on: HETA (Hessian Enhanced Token Attribution),
https://vishalpramanik.github.io/hetaproject.html

What You Are Building

You will build a public web application that explains *why* a language model predicted a specific token. When a user pastes a prompt and selects a target token, your system shows a heatmap highlighting which input tokens most influenced that prediction. The method (HETA) combines attention flow tracing, curvature based sensitivity, and information theoretic masking to produce attributions that are more faithful than simple attention visualization. **You do not need to understand all the math. The prototype handles the core algorithm. Your job is to make it fast, reliable, and usable.**

What Users Will Do

1. Paste a prompt (for example, a question or paragraph)
2. Select a target token position ("Why did it predict THIS word?")
3. See a token heatmap plus a ranked list of most influential tokens
4. Export a JSON report for debugging or audits

Hard Constraints (Non Negotiable)

- **Single GPU with 16GB VRAM or less** (T4, RTX 3060/4060, or similar)
- **Model size approximately 3B parameters** (Qwen2.5 3B recommended; fallback to 1 to 2B if needed)
- **Interactive latency** (seconds, not minutes; define your p95 target in Week 2)
- **Reproducible Docker build** that a TA can run with no manual edits

Your Technical Tasks

A. Integrate the HETA Core Library (provided in prototype)

The prototype implements the three HETA components: semantic transition gate (attention value rollout), curvature term (Hessian estimation), and information gain (KL divergence under masking). You wire these together, handle edge cases (empty input, timeouts, long prompts), and add caching where possible.

B. Make It Fast Enough for a Public Demo

Implement a "quality versus latency" knob (fewer Hessian samples means faster response). Cache tokenization and KV states. Enforce strict input limits. Profile GPU memory and fix any leaks.

C. Build the Frontend and Visualization

Gradio or Streamlit interface with: token heatmap, hover details (score, percentile), JSON/PNG export. Must handle a request queue gracefully.

D. Deploy and Harden: Dockerized GPU service, reverse proxy, rate limiting, telemetry (latency and errors only, no prompt logging).

E. Optional: Quantization Exploration: Investigate running the model in 4 bit or 8 bit quantized form using bitsandbytes or GPTQ. Measure any accuracy degradation in attribution quality versus speedup and memory savings. Consider whether parts of the Hessian estimation can operate on quantized representations.

How You Will Be Evaluated

- **Correctness:** Removing top K attributed tokens reduces target probability more than random removal
- **Performance:** Meet your p95 latency target on a standard prompt length
- **Reliability:** Survive a 2 hour soak test with scripted traffic
- **Usability:** A non expert can get results in under 60 seconds

Team Requirements

Team size: 4 students (minimum); 5 is ideal

Minimum GPA: 3.5 overall and 3.75 in major

Required skills across the team:

- 2 students with PyTorch and Transformers experience (hooks, forward passes, GPU debugging)
- 1 student with web deployment skills (Docker, reverse proxy, rate limiting)
- 1 student with evaluation and benchmarking mindset (datasets, metrics, reproducibility)

One Semester Timeline

Week	Milestone
1 to 3	**Design Review (10%):** Architecture diagram, model choice justified, acceptance tests defined
4 to 6	**Alpha Demo (20%):** End to end works locally in Docker, minimal UI, one eval script runs
7 to 9	**Beta Deployment (30%):** Remote deployment, rate limits, monitoring, evaluation with baselines
10 to 14	**Final Release (40%):** Meets all hard gates, full rubric, final report and recorded demo

Project 2: NEAR Serve

Hallucination Risk Scoring for QA and RAG Applications

Based on: Shapley NEAR (Fact or Hallucination?),
https://vishalpramanik.github.io/NEARproject.html

What You Are Building

You will build a public "fact versus hallucination" service that scores LLM answers. Given a context passage and a question, your system generates an answer and computes a **NEAR score**, a confidence measure based on how much the context actually reduced the model's uncertainty. High score means the answer is grounded in context. Low score means likely hallucination. The system also highlights which sentences in the context contributed most. **The prototype handles the core entropy and Shapley math. Your job is to make it production ready and integrate it into a usable RAG workflow.**

What Users Will Do

1. Provide context plus question (or upload a document to be chunked)
2. See the generated answer plus NEAR confidence score plus "Supported" or "Risky" label
3. View sentence level highlighting showing which context sentences helped most
4. (Optional) Toggle "head clipping" to reduce certain hallucination modes

Hard Constraints (Non Negotiable)

- **Single GPU with 16GB VRAM or less** for the public endpoint
- **Model size approximately 3B parameters** (Qwen2.5 3B primary; optional 6 to 8B in 4 bit for premium mode)
- **Fixed compute budget per request** (no unbounded Shapley permutations)
- **Calibrated threshold** documented and frozen (no hand tuning at demo time)

Your Technical Tasks

A. Integrate the NEAR Scoring Library (provided in prototype)

The prototype extracts attention output norms, computes entropy based information gain (with versus without context), and runs approximate Shapley attribution to sentences. You wire this into a clean API, implement sentence segmentation, and add the optional head clipping toggle.

B. Build a "NEAR for RAG" Reference Pipeline

Chunk documents, retrieve top k passages, generate answer, then compute NEAR score. Output a structured JSON report that downstream apps can consume.

C. Build a Usable Three Panel UI

Panel 1: Answer. Panel 2: NEAR score plus label plus threshold info. Panel 3: Context with sentence highlighting plus top contributors list.

D. Run Calibration and Evaluation: Document your threshold selection procedure. Run evaluation on at least one QA benchmark slice (CoQA, SQuAD, and similar) and report AUROC or correlation metrics.

E. Optional: Quantization Exploration: Test NEAR scoring with 4 bit quantized models. Measure whether entropy and information gain calculations remain accurate under quantization. Document any tradeoffs between memory savings and scoring fidelity.

How You Will Be Evaluated

- **Correctness:** API returns NEAR score plus sentence contributions plus label with documented threshold
- **Validation:** Reproducible evaluation harness on at least one QA dataset with baseline comparison
- **Performance:** Fixed coalition sample budget; p95 latency tracked
- **Integration:** Minimal RAG pipeline that chunks, retrieves, answers, and scores

Team Requirements

Team size: 4 to 5 students

Minimum GPA: 3.5 overall and 3.75 in major

Required skills across the team:

- 2 students comfortable with transformer internals (extracting attention outputs, hooks)
- 1 student comfortable with probability, entropy, and evaluation metrics (AUROC, correlation)
- 1 student comfortable with API engineering and deployment (FastAPI, Docker, GPU hosting)
- *Recommended:* Prior experience with any RAG framework

One Semester Timeline

Week	Milestone
1 to 3	**Design Review (10%):** Requirements spec, threshold calibration plan, model choice, API schema
4 to 6	**Alpha Demo (20%):** NEAR scoring works locally, basic UI, one benchmark slice runs
7 to 9	**Beta Deployment (30%):** RAG pipeline integrated, threshold frozen, deployed with monitoring
10 to 14	**Final Release (40%):** Full evaluation report, soak test passed, documentation complete

Project 3: Nullspace Steering Safety Lab

Responsible Robustness Testing via Mechanism Level Interventions

Based on: HMNS (Head Masked Nullspace Steering), ICLR 2026 submission (provided under NDA)

CYBERSECURITY RESEARCH: This paper describes techniques for testing LLM safety mechanisms. You will NOT build a public attack service. Read the "What You Are NOT Building" section carefully.

What You Are Building

You will build a **two part system** that demonstrates mechanism level steering concepts **safely**. The underlying method (HMNS) identifies which attention heads are most responsible for a model's behavior, masks their outputs, and injects steering signals in the orthogonal nullspace. **Your job is to apply this technique to SAFE tasks (formatting, style, verbosity) and build a dashboard showing robustness insights, NOT to operationalize attacks.**

What You Are NOT Building

- **NO** public endpoint that accepts arbitrary prompts for jailbreak style steering
- **NO** unsafe prompt datasets published or stored without access controls
- **NO** exploit enabling code, scripts, or interfaces released publicly

The Two Part System

Part 1: Gated Lab Mode (restricted to course staff and approved users)

Runs HMNS inspired interventions on a small model using a controlled safe prompt suite. Produces metrics: head importance distributions, steering magnitudes, stability plots. All runs are logged. Authentication required.

Part 2: Public Dashboard Mode (anyone can view)

Shows **aggregate results only**, no unsafe prompts, no raw generations. Includes interactive visualizations of: which heads are causal for safe behaviors (citation formatting, JSON adherence), how nullspace steering changes outputs on benign tasks (style, verbosity).

Hard Constraints (Non Negotiable)

- **Single GPU with 16GB VRAM or less;** model size 1 to 3B parameters
- **Safe task suites ONLY** (JSON formatting, summarization style, citation adherence, no harmful content)
- **Compute normalized metrics** (report internal passes plus latency, not just success counts)
- **Written Responsible Release Plan** reviewed by faculty before any public materials

Your Technical Tasks

A. Implement Mechanism Hooks (core engineering)

Head masking at out projection slices. Residual stream injection. Nullspace direction computation via QR based projector with tolerance checks. The prototype provides reference implementations.

B. Build Safe Task Suites

Examples: JSON schema adherence, faithful summarization versus verbosity, citation formatting, formal and informal style transfer. No harmful or policy violating content.

C. Build Educational Outputs

Per layer and per head causal scores. Steering magnitude versus output divergence plots. Stability across seeds and temperatures. These teach users about model internals.

D. Public Dashboard and Release Controls: Static frontend rendering stored artifacts. Clear documentation. Explicit "what is withheld" statement aligned with cybersecurity ethics.

E. Optional: Quantization Exploration: Investigate whether head masking and nullspace steering work correctly on 4 bit quantized models. Measure if causal attribution rankings change under

quantization. Document any numerical stability issues with QR projections on lower precision tensors.

How You Will Be Evaluated

- **Safety Gate (pass or fail):** No public jailbreak capability. Lab mode gated. This caps grade at C if failed.
- **Correctness:** Mechanism hooks work; nullspace orthogonality verified numerically
- **Robustness Metrics:** Stability across seeds and temperatures on safe tasks; head importance plots; steering magnitude analysis
- **Documentation:** Lab reports for 2 to 3 model variants plus written Responsible Release Plan

Team Requirements

Team size: 5 students (minimum), heavier workload due to infrastructure, safety, and mechanistic code

Minimum GPA: 3.5 overall and 3.75 in major

Required skills across the team:

- 2 students with strong PyTorch internals (hooks, module surgery, attention head slicing)
- 1 student comfortable with numerical linear algebra (QR stability, projections)
- 1 student with MLOps and deployment skills (containers, GPU hosting, monitoring)
- 1 student with responsible AI and evaluation mindset (benchmarking, documentation, release discipline)
- **Non negotiable:** All team members sign a policy forbidding exploit enabling releases; faculty review of all public materials

One Semester Timeline

Week	Milestone
1 to 3	**Design Review (10%):** Safe task suite defined, architecture, Responsible Release Plan draft
4 to 6	**Alpha Demo (20%):** Mechanism hooks working, one safe task end to end, lab mode auth
7 to 9	**Beta Deployment (30%):** Dashboard with aggregate results, 2 or more model variants, nightly pipeline
10 to 14	**Final Release (40%):** Safety gate passed, lab reports, Responsible Release Plan finalized

Document Information

How to Apply

Interested students should contact Dr. Sumit Kumar Jha with: (1) a resume or CV, (2) a brief statement of interest (500 words) explaining relevant experience and which project they prefer, and (3) a link to any relevant code repositories or prior projects. **Teams may be formed from individual applicants with complementary skills.**

Resources and References

Project 1: HETA (Hessian Enhanced Token Attribution)

https://vishalpramanik.github.io/hetaproject.html

Project 2: Shapley NEAR (Fact or Hallucination?)

https://vishalpramanik.github.io/NEARproject.html

Project 3: HMNS (Head Masked Nullspace Steering) (AAAI 2026 workshop paper; available soon)

Hardware Access

Students may have access to departmental GPU resources for experiments requiring larger models. Consumer-grade hardware (16GB RTX 30, 40, or 50 series) is sufficient for development with quantized 3B models. Google Colab Pro may also be adequate. The faculty will endorse GPU requests to the department or others but will not be responsible for GPU access.

Acknowledgment

This document was edited using multiple generative AI models.